

AMENDMENT(S) TO THE CLAIMS

1  
2  
3  
4       1.     (currently amended)   A method for designing an application  
5 programming interface (API), the method comprising:

6           preparing a plurality of code samples for a core scenario, each respective  
7 code sample of the plurality of code samples corresponding to a respective  
8 programming language of a plurality of programming languages; ~~and~~

9           deriving the API from the core scenario responsive to the plurality of code  
10 samples; and

11           realizing the derived API in one or more processor-accessible storage  
12 media.

13  
14       2.     (currently amended)   The method as recited in claim 1, further  
15 comprising:

16           determining, by an API designer, if the derived API is ~~too~~ more complex  
17 than desired.

18  
19       3.     (currently amended)   The method as recited in claim 2, further  
20 comprising:

21           if the derived API is determined to be ~~too~~ more complex than desired, then  
22           refining, by the API designer, the derived API to produce a refined  
23 API.  
24  
25

1           4.     (currently amended) The method as recited in claim 3, further  
2 comprising:

3                 determining, by the API designer, if the refined API is ~~too~~more complex  
4 than desired.

5  
6           5.     (original) The method as recited in claim 1, further comprising:  
7 performing one or more usability studies on the API utilizing a plurality of  
8 developers.

9  
10          6.     (currently amended) The method as recited in claim 5, wherein the  
11 performing comprises:

12                 performing the one or more usability studies on the API utilizing the  
13 plurality of developers wherein the plurality of developers are ~~typical for~~  
14 competent with the plurality of programming languages.

15  
16          7.     (currently amended) The method as recited in claim 5, further  
17 comprising:

18                 ascertaining whether the plurality of developers are able to use the API  
19 without ~~significant~~ problems.

20  
21          8.     (currently amended) The method as recited in claim 7, further  
22 comprising:

23                 if the plurality of developers are not ascertained to be able to use the API  
24 without ~~significant~~ problems, then revising the API.

1           **9.**     (original) The method as recited in claim 8, wherein the revising  
2 comprises:

3                   revising the API based on at least one lesson from the one or more  
4 usability studies.

5  
6           **10.**   (original) The method as recited in claim 1, wherein the deriving  
7 comprises:

8                   deriving the API to support the plurality of code samples that  
9 correspond respectively to the plurality of programming languages.

10  
11          **11.**   (original) The method as recited in claim 1, wherein the deriving  
12 comprises:

13                   gleaning language-specific mandates from the plurality of code  
14 samples; and  
15                   incorporating the language-specific mandates into the API.

16  
17          **12.**   (original) The method as recited in claim 1, wherein the deriving  
18 comprises:

19                   gleaning language-inspired developer expectations from the plurality  
20 of code samples; and  
21                   incorporating the language-inspired developer expectations into the  
22 API.

1       13. (original) The method as recited in claim 1, wherein the deriving  
2 comprises:

3               gleaning commonalities from the plurality of code samples; and  
4               incorporating the commonalities into the API.  
5

6       14. (original) The method as recited in claim 1, wherein the deriving  
7 comprises:

8               deriving the API to have an aggregate component that ties a plurality  
9 of lower-level factored types together to support the core scenario.  
10

11       15. (currently amended) A method for designing an application  
12 programming interface (API), the method comprising:

13               selecting a core scenario for a feature area;

14               writing at least one code sample for the core scenario; and

15               deriving an API for the core scenario responsive to the at least one code  
16 sample; and

17               realizing the derived API in one or more processor-accessible storage  
18 media.  
19

20       16. (original) The method as recited in claim 15, wherein the selecting  
21 comprises:

22               selecting a plurality of core scenarios for the feature area.  
23  
24  
25

1           **17.**   (original) The method as recited in claim 16, further comprising:  
2           repeating the writing and the deriving for each core scenario of the plurality  
3 of core scenarios that are selected for the feature area.

4  
5           **18.**   (original) The method as recited in claim 15, wherein the writing  
6 comprises:

7                   writing a plurality of code samples for the core scenario, each  
8                   respective code sample of the plurality of code samples corresponding to a  
9                   respective programming language of a plurality of programming languages.

10  
11           **19.**   (original) The method as recited in claim 18, wherein the deriving  
12 comprises:

13                   deriving the API for the core scenario responsive to the plurality of  
14                   code samples.

15  
16           **20.**   (currently amended) The method as recited in claim 15, further  
17 comprising:

18                   performing one or more usability studies on the API utilizing a plurality of  
19 developers;

20                   ascertaining whether the plurality of developers are able to use the API  
21 without significant problems; and

22                   if the plurality of developers are not ascertained to be able to use the API  
23 without significant problems, then revising the API.

1           **21.**   (original) The method as recited in claim 20, wherein the revising  
2 comprises:

3                   revising the API based on at least one lesson from the one or more  
4 usability studies to produce a revised API.

5  
6           **22.**   (original) The method as recited in claim 21, further comprising:  
7 repeating the performing and the ascertaining with respect to the revised  
8 API.

9  
10          **23.**   (original) The method as recited in claim 15, wherein the deriving  
11 comprises:

12                   deriving the API to support the at least one code sample written for  
13 the core scenario by producing a two-layer API that includes an aggregate  
14 component and a plurality of underlying factored types.

15  
16          **24.**   (original) The method as recited in claim 15, wherein the deriving  
17 comprises:

18                   gleaning one or more language-specific mandates from the at least  
19 one code sample; and

20                   incorporating the one or more language-specific mandates into the  
21 API.

1       **25.**   (original) The method as recited in claim 15, wherein the deriving  
2 comprises:

3               encapsulating a particular factored type into an aggregate component  
4 that is associated with the core scenario if all members of the particular  
5 factored type are exposed by the aggregate component.  
6

7       **26.**   (currently amended) The method as recited in claim 15, wherein the  
8 deriving comprises:

9               encapsulating a particular factored type into an aggregate component  
10 that is associated with the core scenario if the particular factored type is  
11 independently ~~uninteresting~~ unrelated to other component types.  
12

13       **27.**   (original) The method as recited in claim 15, wherein the deriving  
14 comprises:

15               exposing a particular factored type from an aggregate component  
16 that is associated with the core scenario if at least one member of the  
17 particular factored type is not exposed by the aggregate component.  
18

19       **28.**   (original) The method as recited in claim 15, wherein the deriving  
20 comprises:

21               exposing a particular factored type from an aggregate component  
22 that is associated with the core scenario if the particular factored type can  
23 be beneficially used independently of the aggregate component by another  
24 component type.  
25

1       **29.**   (original) The method as recited in claim 15, wherein the deriving  
2 comprises:

3               producing a two-layer framework that includes component types  
4               targeting a relatively higher level of abstraction and component types  
5               targeting a relatively lower level of abstraction.

6  
7       **30.**   (original) The method as recited in claim 29, wherein the component  
8 types targeting the relatively higher level of abstraction are directed to core  
9 scenarios.

10  
11       **31.**   (original) The method as recited in claim 29, wherein the component  
12 types targeting the relatively lower level of abstraction provide a relatively greater  
13 amount of control to developers as compared to the component types targeting the  
14 relatively higher level of abstraction.

15  
16       **32.**   (original) The method as recited in claim 15, wherein the deriving  
17 comprises:

18               deriving the API so as to enable a developer to implement a create-  
19 set-call usage pattern for the core scenario.

20  
21       **33.**   (original) The method as recited in claim 32, wherein the deriving  
22 comprises:

23               producing the API with pre-selected parameters that are  
24               appropriate for the core scenario.



1       **34.**   (original) A method for designing an application programming  
2 interface (API), the method comprising:

3       deriving an API for a scenario responsive to at least one code sample  
4 written with regard to the scenario;

5       performing one or more usability studies on the API utilizing a plurality of  
6 developers; and

7       revising the API based on the one or more usability studies.

8  
9       **35.**   (original) The method as recited in claim 34, further comprising:

10      writing a plurality of code samples with regard to the scenario, each  
11 respective code sample of the plurality of code samples corresponding to a  
12 respective programming language of a plurality of programming languages;

13      wherein the deriving comprises:

14          deriving the API for the scenario responsive to the plurality of code  
15 samples.

1           **36.**   (currently amended) The method as recited in claim 34, further  
2 comprising, prior to the performing one or more usability studies on the API:

3           determining, by an API designer, if the derived API is ~~too~~more complex  
4 than desired;

5           if the derived API is determined to be ~~too~~more complex than desired, then

6           refining, by the API designer, the derived API to produce a refined  
7 API; and

8           determining, by the API designer, if the refined API is ~~too~~more  
9 complex than desired.

10  
11           **37.**   (currently amended) The method as recited in claim 34, further  
12 comprising:

13           ascertaining whether the plurality of developers are able to use the API  
14 without ~~significant~~ problems; and

15           when the plurality of developers are not ascertained to be able to use the  
16 API without ~~significant~~ problems, then implementing the revising;

17           wherein the revising comprises:

18           revising the API based on at least one lesson from the one or more  
19 usability studies to produce a revised API.

20  
21           **38.**   (original) The method as recited in claim 37, further comprising:  
22           repeating at least the performing and the ascertaining with respect to the  
23 revised API.  
24  
25

1           **39.**   (currently amended) The method as recited in claim 37, wherein the  
2   ascertaining comprises:

3                   ascertaining whether the plurality of developers are able to use the  
4           API without ~~significant~~ problems with regard to a desired level of usability  
5           for at least one targeted developer group, wherein the desired level of  
6           usability includes considerations with respect to (i) frequent and/or  
7           extensive reference to detailed API documentation by the plurality of  
8           developers, (ii) a failure of a majority of the plurality of developers to  
9           implement the scenario, and (iii) whether the plurality of developers take an  
10          approach that is ~~significantly~~ different from what is expected by an API  
11          designer.

12  
13           **40.**   (currently amended) The method as recited in claim 34, further  
14   comprising:

15                   selecting a plurality of core scenarios for a feature area; and  
16                   repeating the deriving, the performing, and the revising for each core  
17          scenario of the plurality of core scenarios.

18  
19           **41.**   (original) The method as recited in claim 40, wherein the deriving  
20   comprises:

21                   producing an aggregate component for each core scenario of the  
22          plurality of core scenarios.

1           **42.**   (original) The method as recited in claim 34, wherein the deriving  
2 comprises:

3                   producing an aggregate component that has a respective relationship  
4 with each respective factored type of a plurality of factored types.

5  
6           **43.**   (original) The method as recited in claim 42, wherein the producing  
7 comprises:

8                   producing the aggregate component to support the scenario  
9 for which the at least one code sample is written.

10  
11          **44.**   (original) The method as recited in claim 42, wherein the producing  
12 comprises:

13                   producing the aggregate component to have an exposed  
14 relationship with at least one factored type of the plurality of  
15 factored types and an encapsulated relationship with at least one  
16 other factored type of the plurality of factored types.

17  
18          **45.**   (original) The method as recited in claim 44, wherein the at least  
19 one other factored type of the plurality of factored types that has the encapsulated  
20 relationship with the aggregate component can be handed off by the aggregate  
21 component for direct interaction with another component type.

22  
23          **46.**   (original) The method as recited in claim 42, wherein the plurality of  
24 factored types are designed using an object-oriented methodology.  
25

1           47. (original) A method for designing an application programming  
2 interface (API), the method comprising:

3           preparing a plurality of code samples for a core scenario, each respective  
4 code sample of the plurality of code samples corresponding to a respective  
5 programming language of a plurality of programming languages;

6           deriving the API for the core scenario responsive to the plurality of code  
7 samples;

8           performing one or more usability studies on the API utilizing a plurality of  
9 developers; and

10          revising the API based on the one or more usability studies.

11  
12          48. (currently amended) A method for designing an application  
13 programming interface (API), the method comprising:

14          writing at least one code sample for a scenario; ~~and~~

15          deriving an API for the scenario responsive to the at least one code sample,  
16 the API including (i) an aggregate component that is adapted to facilitate  
17 implementation of the scenario and (ii) a plurality of factored types that provide  
18 underlying functionality for the aggregate component, the API enabling a ~~gradual~~  
19 progression from using the aggregate component in simpler situations to using an  
20 increasing portion of the plurality of factored types in increasingly complex  
21 situations; and

22          realizing the derived API in one or more processor-accessible storage  
23 media.

1        49. (currently amended) A method for designing an application  
2 programming interface (API), the method comprising:

3        deriving at least one aggregate component to support at least one code  
4 sample for at least one scenario;

5        determining additional requirements with respect to the at least one  
6 scenario;

7        deciding if the additional requirements can be added to the at least one  
8 aggregate component without adding ~~undue~~ more complexity than desired to the at  
9 least one scenario; ~~and~~

10        if it is decided that the additional requirements can not be added to the at  
11 least one aggregate component without adding more complexity than desired to the  
12 at least one scenario, then:

13                defining a plurality of factored types responsive to the deciding;

14                realizing the at least one aggregate component in one or more  
15 processor-accessible storage media; and

16                realizing the plurality of factored types in the one or more processor-  
17 accessible storage media; and

18        if it is decided that the additional requirements can be added to the at least  
19 one aggregate component without adding more complexity than desired to the at  
20 least one scenario, then:

21                refining the at least one aggregate component to incorporate the  
22 additional requirements; and

23                realizing the refined at least one aggregate component in the one or  
24 more processor-accessible storage media.  
25

1       **50.**   (canceled)

2  
3       **51.**   (original) The method as recited in claim 49, further comprising:  
4       selecting a plurality of core scenarios for a feature area, the plurality of core  
5       scenarios including the at least one scenario; and  
6       writing a plurality of code samples showing preferred lines of code for the  
7       plurality of core scenarios, the plurality of code samples including the at least one  
8       code sample;

9       wherein the deriving comprises:

10       deriving a plurality of aggregate components, which include the at  
11       least one aggregate component, to support the plurality of code samples for  
12       the plurality of core scenarios.

13  
14       **52.**   (currently amended) The method as recited in claim 49, wherein the  
15       deriving comprises:

16       deriving the at least one aggregate component with appropriate  
17       methods, defaults, and abstractions to support the at least one code sample  
18       for the at least one scenario.

1           **53.**   (currently amended) The method as recited in claim 49, further  
2 comprising:

3           refining the at least one code sample according to the at least one derived  
4 aggregate component;~~and~~

5           evaluating the refined at least one code sample with regard to simplicity;  
6 and

7           repeating the deriving if the refined at least one code sample fails to be  
8 ~~sufficiently as simple as desired as determined~~ in the evaluating.

9  
10          **54.**   (original) The method as recited in claim 49, wherein the  
11 determining comprises:

12           determining additional requirements with respect to the at least one  
13 scenario, wherein the additional requirements include additional scenarios,  
14 additional usages, and additional interactions with other component types.

15  
16          **55.**   (original) The method as recited in claim 49, wherein the deciding  
17 comprises:

18           considering whether adding the additional requirements to the at  
19 least one aggregate component hinders a create-set-call usage pattern.

20  
21          **56.**   (currently amended) The method as recited in claim 49, wherein the  
22 defining comprises:

23           defining the plurality of factored types responsive to the deciding  
24 with ~~an ideal a~~ factoring of a full set of functionality.



1           **57.**   (original) The method as recited in claim 49, wherein the defining  
2 comprises:

3                 defining the plurality of factored types responsive to the deciding  
4                 using one or more object-oriented methodologies.

5  
6           **58.**   (original) The method as recited in claim 49, further comprising:  
7                 determining whether the at least one aggregate component is to encapsulate  
8                 or expose the functionality of each factored type of the plurality of factored types.

9  
10          **59.**   (original) The method as recited in claim 49, further comprising:  
11                 refining the plurality of factored types to support the at least one aggregate  
12                 component and the additional requirements.